1) **Word Break Problem**

def word\_break\_recursive(s, word\_dict):

if not s:

return True

for word in word\_dict:

if s.startswith(word):

if word\_break\_recursive(s[len(word):], word\_dict):

return True

return False

dictionary = {"i", "like", "sam", "sung", "samsung", "mobile", "ice", "cream", "icecream", "man", "go", "mango"}

input\_string1 = "ilike"

input\_string2 = "ilikesamsung"

print("Input:", input\_string1, "Output:", "Yes" if word\_break\_recursive(input\_string1, dictionary) else "No")

print("Input:", input\_string2, "Output:", "Yes" if word\_break\_recursive(input\_string2, dictionary) else "No")

Input: ilike

Output: Yes

Input: ilikesamsung

Output: Yes

2) **Word Wrap Problem**

def fullJustify(words, maxWidth):

res, current\_line, num\_of\_letters = [], [], 0

for word in words:

if num\_of\_letters + len(word) + len(current\_line) > maxWidth:

for i in range(maxWidth - num\_of\_letters):

current\_line[i % (len(current\_line) - 1 or 1)] += ' '

res.append(''.join(current\_line))

current\_line, num\_of\_letters = [], 0

current\_line.append(word)

num\_of\_letters += len(word)

res.append(' '.join(current\_line).ljust(maxWidth))

return res

print(fullJustify(["This", "is", "an", "example", "of", "text", "justification."], 16))

Input: words = ["This", "is", "an", "example", "of", "text", "justification."], maxWidth = 16

Output:

[ "This is an",

"example of text",

"justification. "

]

Input: words = ["What","must","be","acknowledgment","shall","be"], maxWidth = 16

Output:

[

"What must be",

"acknowledgment ",

"shall be "

]

3) **Word Wrap Problem**

class WordFilter:

def \_\_init\_\_(self, words):

self.words = words

def f(self, pref, suff):

max\_index = -1

for i, word in enumerate(self.words):

if word.startswith(pref) and word.endswith(suff):

max\_index = max(max\_index, i)

return max\_index

Input

["WordFilter", "f"]

[[["apple"]], ["a", "e"]]

Output

[null, 0]

4) **Warshall’s & Floyd’s Algorithm**

def unique\_paths(m: int, n: int) -> int:

dp = [[1] \* n for \_ in range(m)]

for i in range(1, m):

for j in range(1, n):

dp[i][j] = dp[i - 1][j] + dp[i][j - 1]

return dp[m - 1][n - 1]

print(unique\_paths(3, 7))

print(unique\_paths(3, 2))

Input: m = 3, n = 7

Output: 28

Input: m = 3, n = 2

Output: 3

5)**Good Pair**

A pair (i, j) is called good if nums[i] == nums[j] and i < j.

def count\_good\_pairs(nums):

count = 0

n = len(nums)

for i in range(n):

for j in range(i + 1, n):

if nums[i] == nums[j]:

count += 1

return count

print(count\_good\_pairs([1, 2, 3, 1, 1, 3]))

print(count\_good\_pairs([1, 1, 1, 1]))

print(count\_good\_pairs([1, 2, 3]))

Input: nums = [1,2,3,1,1,3]

Output: 4

Input: nums = [1,1,1,1]

Output: 6

Input: nums = [1,2,3]

Output: 0